Application Control

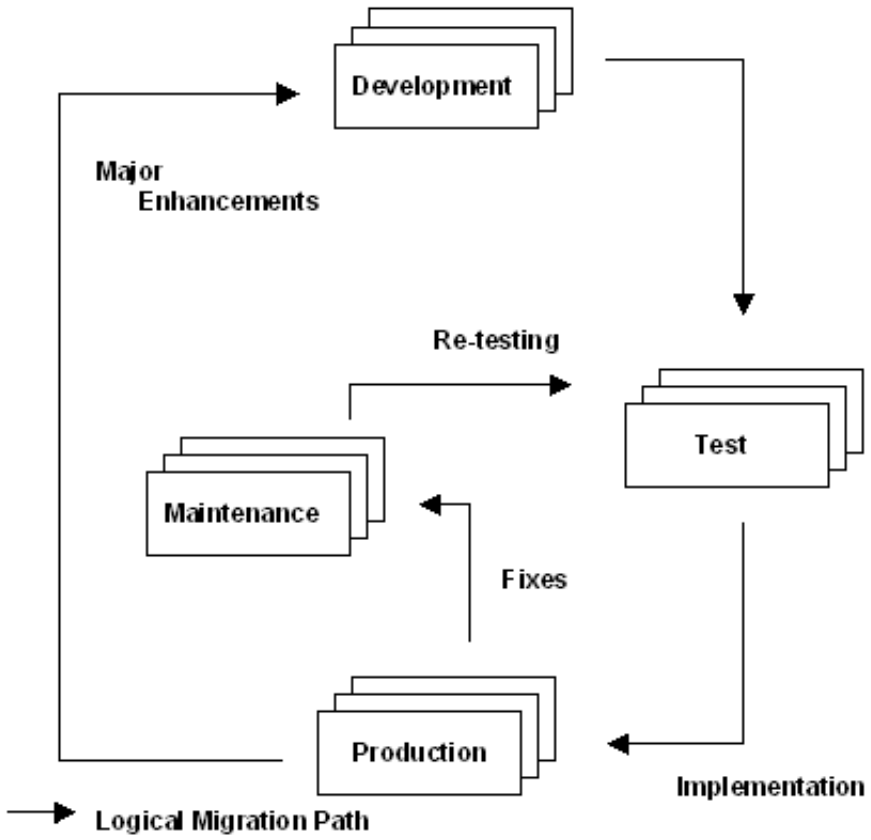
Data processing applications and user environments are continually becoming more complex:

- Multiple hardware and operating system configurations frequently coexist in the same facility.
- Multiple programs may access the same routine in an application library or across applications.
- Several programmers may change the same code concurrently.
- Different versions of a program or routine often exist in development, test, and production libraries.

This increased complexity has multiplied the problems of controlling an application throughout its life-cycle.

- Inventory Management
- Configuration Management
- Migration Management
- Production Management
- Change Management

The following figure shows an example of an application life-cycle in a heterogeneous user environment.



Application control issues can be broken down into five areas of software management:

1. Inventory
2. Configuration
3. Migration
4. Production
5. Change

The following sections present the questions and processes involved in each area and explain how PAC addresses them.

Inventory Management

This section answers the following questions:

- Where is the source code for an object?
- Do the executable objects match the source code?
- What is the current version of an object?
- What other objects does an object need for successful compilation and execution?
- Which version of an object does a calling program use?
- How are obsolete objects handled?

Managers and auditors increasingly cite the need for better control over source and executable code. Inventory management maintains the source and executable code for all components of an organization’s applications. It maintains the source and executable libraries, links executable objects to source code, and provides an efficient way to retrieve the source code.

Each time an object is migrated from a development or maintenance environment to another environment defined to PAC (for example, for testing), PAC compiles the object, assigns it a new version number, and stores both the source and executable code in a protected file. PAC uses a date-time stamp to synchronize the source and executable code for each version of an object, or object version. This mechanism happens automatically for Natural objects, but has to be controlled using supplied APIs for foreign objects.

A reporting facility shows all versions of an object in the PAC-controlled environment, including the current version. For each object version, you can display the source code, directory information about the operating environment in which the source code was developed, and other information.

When the relationships among versions of component objects are not up to date, a new version of an application may fail in production. Inventory management maintains accurate relationships among versions of interdependent objects.

PAC automatically cross-references object versions. Thus, not only can you list all the objects that an object uses or all objects that use the object, but also you can list the specific object versions. This is only possible for Natural objects.

To help you keep track of object versions, PAC enables you to recompile an entire application using the latest versions in the PAC-controlled environment or other versions that you specify.

PAC includes two facilities for handling obsolete objects. When you archive objects, you can unload them to disk or tape and purge them from the PAC-controlled environment, if necessary, you can later restore them to PAC, this is called archiving. When you retire objects, you purge them from the PAC-controlled environment by migrating them to a "virtual" environment; objects may or may not be archived before being retired.

Configuration Management

This section answers the following questions:

- How are the development, test, production, and maintenance environments defined?
- What operating environments are defined for a particular application?
- What is the physical location of a particular application environment?
- What is the life-cycle of the application?
- Who can authorize migrations of objects between two environments?

Configuration management establishes the operating environments for applications. For each environment, it specifies the physical location, how objects are handled, and what applications may use it.

For each application, configuration management defines all the environments the application may use during its life-cycle, the valid migration paths among them, and the controls on those paths. The structure of permitted environments and migration paths constitutes the application life-cycle.

In PAC, you can define the attributes of an application environment once and enforce them for all users. PAC allows you to define multiple development, test, production, and maintenance environments. Each application is assigned to a subset of the environments; applications can share environments.

Managers can control who may define the valid migration paths among each application’s environments. Each migration-path definition includes controls on migrations along that path, including who may authorize the migration. A configuration of migration paths can be copied from one application to another.

Thus, PAC provides both central control and flexibility. Managers can establish the degree of central control in configuring operating environments and authorizing migrations among them. Depending on the needs and philosophy of your organization, you can create varied or uniform application life-cycles.

Migration Management

This section answers the following questions:

- How do you handle migrations when the origin and destination environments use different operating systems, teleprocessing (TP) systems, database management systems (DBMS’s), and databases?
- How are procedures for migrating or promoting objects enforced?
- How do you ensure that the correct object version is implemented and referenced by other objects?
- How do you recover when an object is accidentally overlaid?
- How do you handle a situation in which an object was compiled referencing one set of databases and files and you need to execute it referencing a different set?

Control over the migration and promotion of applications through the life-cycle is essential to application integrity. Migration management includes controls to ensure that objects are migrated only to approved locations and that appropriate personnel authorize all migrations. It provides a way to recover from accidental overlays and an accurate

audit trail of all migration activities.

When you migrate an object, migration management helps ensure that all objects needed to compile or execute the object are migrated also. For example, assume that Program (A) uses Subroutine (B) during execution; if Program (A) is migrated to a new location for testing, there must be a way to ensure that Subroutine (B) is migrated with it.

PAC operates independently of the operating system, TP system, and DBMS. Thus, you use the same PAC procedures to migrate applications among heterogeneous environments.

Your definitions for the possible phases of an application's life-cycle, the physical location of each phase and the valid paths for migrating objects to each phase enforce procedures. PAC migrates objects only in accordance with your site- or application-specific definitions.

When you migrate or promote an application to another phase in the life-cycle, PAC uses the latest object versions in the library by default; you can override the default and specify a different version of an object. You can display the directory or source code for any object version or compare versions of an object.

A protected PAC system file stores a copy of every version of every object in the PAC-controlled environment. If one of these versions is accidentally overlaid in a library, it can be reloaded. In addition, when you replace a production object with a new version, PAA automatically backs up the old version; you can restore the earlier version from the backup.

PAC file translation tables save time and computer resources by allowing you to execute compiled objects against different files and databases without changing and recompiling the source code. When an object referencing user data on one set of databases and files is migrated for execution referencing user data on a different set, PAC dynamically recompiles the object to reference the new database and file numbers.

PAC includes a migration utility with several modes for selecting objects. PAC maintains an audit trail of all migration activities.

Production Management

This section answers the following questions:

- How is implementation of application objects controlled?
- When there are multiple production environments, how do you know where objects are implemented?
- How is implementation tracked?
- What facilities exist for recovery when a newly implemented object causes problems in the production environment?

The ultimate goal of any application-control system is to ensure the integrity of applications in the production environment. An organization must be able to protect its production code. Production management protects, audits, and controls objects in production environments.

Although inventory, configuration, and migration management help ensure that correct and tested versions of objects are implemented into production, an application may still cause a system failure when implemented. When this happens, there must be a way to back out quickly to an earlier version of the application or objects.

Predict Application Audit (PAA) protects each production environment and audits all migrations of objects into the environment. When an application is migrated to production, PAA backs up the existing production version of any object that will be replaced by a new version. The PAA administrator can back out a migration, which automatically restores the previous versions of objects that were replaced by the migration.

PAC and PAA also enable you to migrate applications to production in advance and schedule the earliest date and time that they can be activated. The PAA administrator controls the activation.

PAA provides extensive reporting facilities. You can display information about production jobs, applications, libraries, and object backups. You can list all the applications in a library or all the libraries that contain objects of a specified application.

Auditors will appreciate the ease with which changes to production applications (for example, when versions of objects are superseded) can be detailed. Audit reports provide information about each object version and its migration to the production environment. PAA also supply reporting APIs for the end-user's use.

Change Management

This section answers the following questions:

- How are maintenance activities tracked and controlled?
- Who is changing an object?
- How are concurrent changes handled?
- How are related problems grouped?

Change management coordinates the maintenance activities of programmers, ensures that critical problems are resolved quickly, and minimizes the time required. It includes facilities to do the following:

- control updates to code;
- enforce standard maintenance procedures;
- track maintenance requests and activities;
- report the status or history of changes;
- analyze the impact of proposed code changes.

In PAC, when an object is migrated to a maintenance environment, a check out facility automatically logs detailed information about the migration, including the identity of the user who checked the object out. PAC logs the same information when the object is checked in (migrated from the maintenance environment). Users can display or print the logs. PAC controls migration to maintenance environments the way it controls migrations or promotions to any phase of the life-cycle: objects can be migrated only in authorized migrations along defined paths.

You can monitor and prohibit concurrent maintenance activities; by default, PAC handles them with a warning. For example, assume that Programmer A has checked Version 1 of the DISPLAY-ORDER subroutine out for maintenance and has not yet checked it back in; when Programmer B checks out this version of the subroutine, PAC issues a warning to Programmer B. Alternatively, the PAC administrator can monitor the check-outs and block Programmer B from checking out Version 1 until Programmer A checks it back in.

Each time an object is checked back in from a maintenance environment, PAC gives it a new version number. Using the previous example, assume that Programmers A and B concurrently check out Version 1 to maintenance. Programmer A checks the subroutine back in; PAC compiles it and designates it Version 2. Then Programmer B checks the subroutine in; PAC compiles it and designates it Version 3. A PAC utility enables you to compare the two versions.

You can use PAC maintenance requests to document problems with objects, track maintenance activities performed in response to a request, and group related problems and activities. A maintenance request can be linked to an external problem-tracking system.

PAC supports impact analysis by listing all application objects that will be affected by changes to an object.

